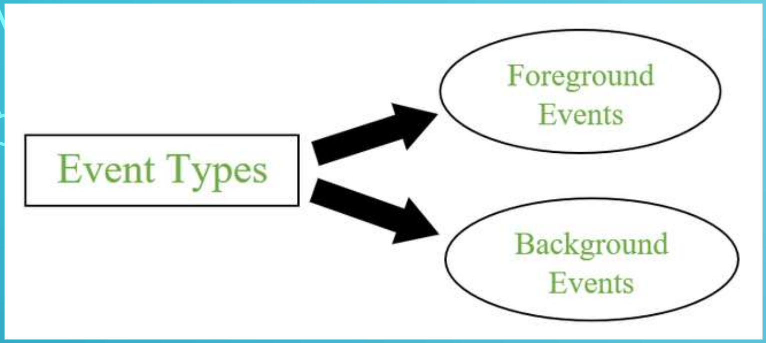
EVENT HANDLING IN JAVA

- Change in the state of an object is known as event i.e. event describes the change in state of source.
- Events are generated as result of user interaction with the graphical user interface components.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

CLASSIFICATION OF EVENTS



Foreground Events –

- Those events which require the direct interaction of user.
- They are generated as consequences of a person interacting with the graphical components in Graphical User Interface.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

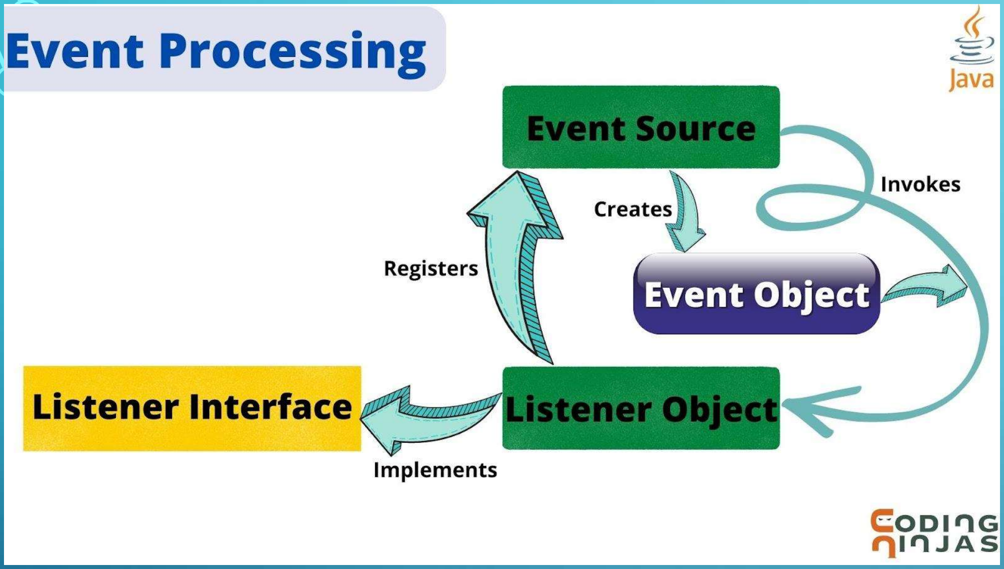
Background Events –

- Those events that require the interaction of end user are known as background events.
- Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

- Event handling is the process to monitor any change in the state of an object and perform a corresponding action.
- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism have the code which is known as event handler that is executed when an event occurs.
- Java Uses the **Event Delegation Model** to handle the events.
- This model defines the standard mechanism to generate and handle the events.

EVENT DELEGATION MODEL

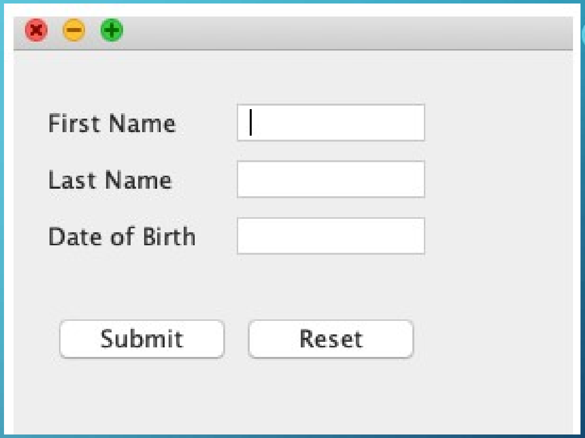
- The modern approach to handling events are based on the delegation model, which defines standard and consistent mechanism to generate and process events.
- The event delegation is made up of 3 components, which are,
  - **Source** - helps to generate an event
  - **Events** – describes the changes in the so that is, it generated by the source.
  - **Event Listener** – receives and processing the events.



- A source generates an event when the state of an object change.
- Event listener is an object that waits for an event to occur.
- When an event is triggered an event listener is notified by the source to which it is registered.

Sources of Event

- An event is generated, when user interacts with sources.
- AWT and Swing are used to develop window-based applications in Java.
- AWT is Abstract Window Toolkit that provides various component classes like Label, Button, TextField, etc., to show window components on the screen.
- All these classes are part of the Java.awt package.



- On the other hand, Swing is the part of JFC (Java Foundation Classes) built on the top of AWT and written entirely in Java.
- The javax.swing API provides all the component classes like JButton, JTextField, JCheckbox, JMenu, etc.

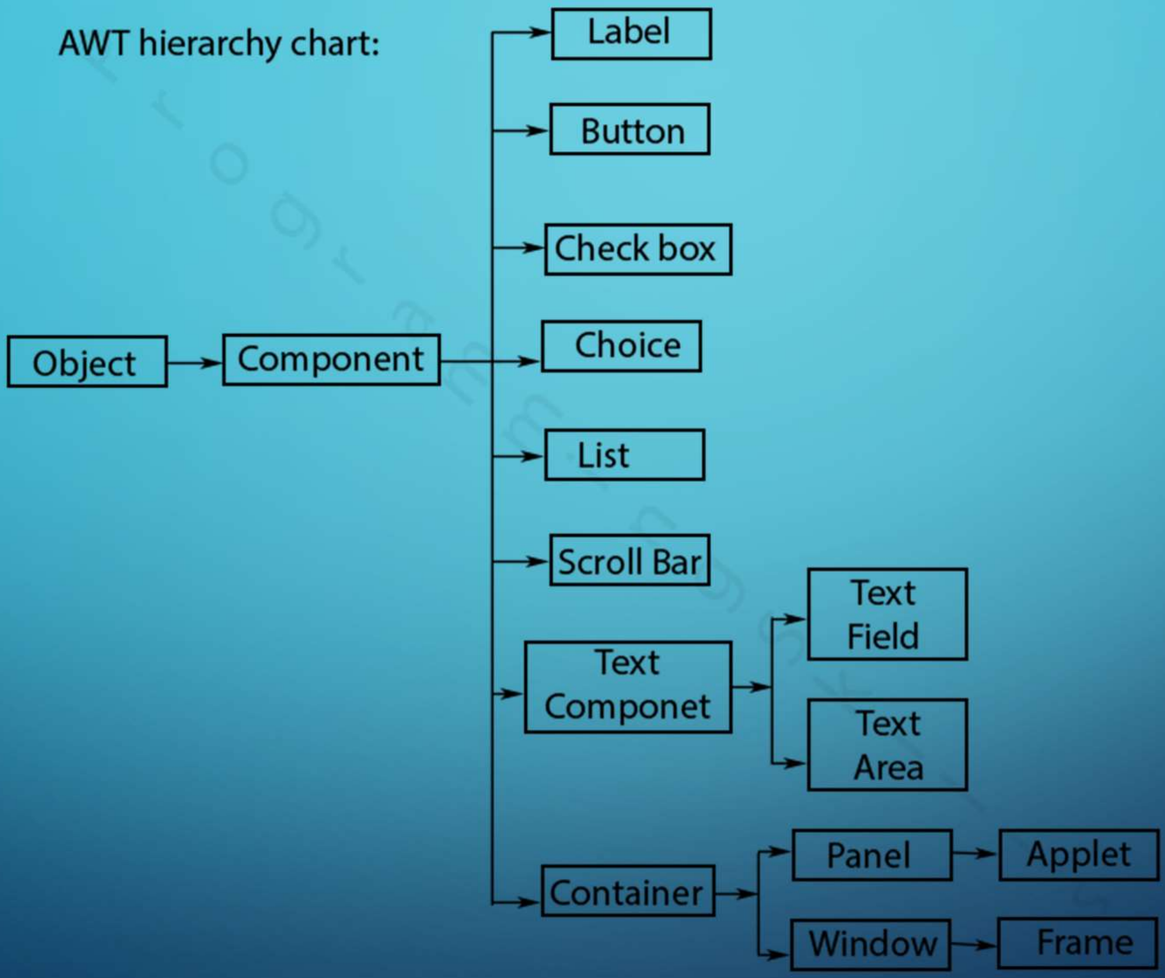
Difference between AWT and Swing

Java AWT	Java Swing
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

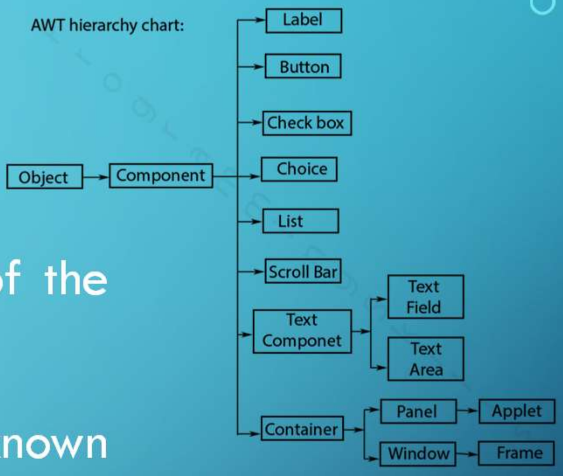
Abstract Window Tool Kit (AWT)

- The most important package in Java are AWT packages, which contains several classes and interfaces.
- The java.awt contains classes that can be classified into 3 groups.
  - Component class
  - Container class
  - Helper class

AWT hierarchy chart:



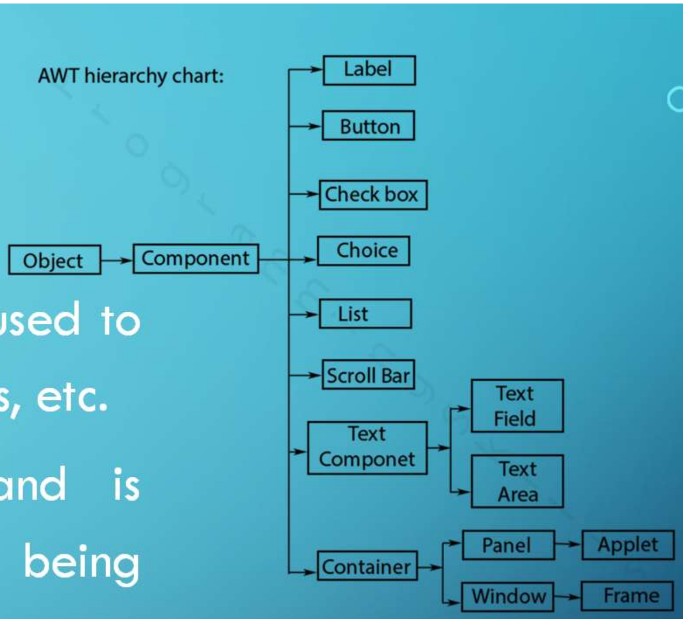
- Component class
  - At the top of the AWT hierarchy is the Component class.
  - Component is an abstract class that encapsulates all of the attributes of visual components.
  - AWT components such as Label, Button, etc are also known as control fundamentals that allows the user to interact with the application.
  - That is all the interface elements that are displayed on the screen and that interact with user are subclass of Component class.



Container class

- Container in Java AWT is a component that is used to hold other components such as text fields, buttons, etc.
- It is a subclass of java.awt.Component and is responsible for keeping a track of components being added.
- There are 2 main subclasses of containers provided by AWT in Java.
  1. Window
  2. Panel

AWT hierarchy chart:



1. Window

- Window class creates a top-level window, with no boarders or title.
- **Frame:** Frame is a subclass of Window and contains title, border and menu bars.
  - It comes with a resizing canvas and is the most widely used container for developing AWT applications.
  - It is capable of holding various components such as buttons, text fields, scrollbars, etc.
  - We can create a Java AWT Frame in two ways:
    - By Instantiating Frame class
    - By extending Frame class
- **Dialog:** Dialog class is also a subclass of Window and comes with the border as well as the title.
  - Dialog class's instance always needs an associated Frame class instance to exist.

2. Panel

- Panel does not contain title bar, menu bar or border.
- It is a generic container for holding components.
- An instance of the Panel class provides a container to which to add components.

- Helper class
  - Graphics, Font, Color etc are known as helper classes
  - Such classes are used to support GUI components.

The various AWT components, which act as an event source is given below,

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected
Scrollbar	Generates adjustment events when the scroll bar is manipulated
Text components	Generates text events when the user enters a character.
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

WINDOW AND FRAME CLASSES

- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.
- AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT- Components such as Button, TextField, Label, TextArea, List, Choice, RadioButton, CheckBox, etc.

- Although creating applets is a common use for Java’s AWT- based Applet, it is also possible to create standalone AWT-based applications.
- To do this, simply create an instance of the window or windows we need inside main( ).

- The **Window** class creates a top-level window.
- A *top-level window* is not contained within any other object; it sits directly on the desktop.
- We won’t create **Window** objects directly.
- Instead, we will use a subclass of **Window** called **Frame.**
- **Frame** encapsulates what is commonly thought of as a “window.”
- It is a subclass of **Window** and has a title bar, menu bar, borders, and resizing corners.
- The precise look of a **Frame** will differ among environments.

- The Frame class is a subclass of Window that encapsulates an application window.
- A Frame object is capable of containing a menu bar and displaying a title.
- Following are two of Frame’s constructors:
  - Frame( ) - creates a standard window that does not contain a title
  - Frame(String title)-creates a window with the title specified by title.

- It is not possible to specify the dimensions of the window while creating it.
- Instead, we must set the size of the window after it has been created.
- void setSize(int newWidth, int newHeight)-
  - Set the size of window specified by newWidth and newHeight.
- void setVisible(boolean visibleFlag) –
  - make the window visible if visibleFlag is true. Otherwise, it is hidden.
- void setTitle(String newTitle) –
  - to change the title in a frame window where newTitle is the new title for the window.

- When working with Frame objects, there are basically three steps involved to get a Frame window to appear on the screen:
  - Instantiate the Frame object
  - Give the Frame object a size using setSize()
  - Make the Frame appear on the screen by invoking setVisible(true).

Creating a Frame

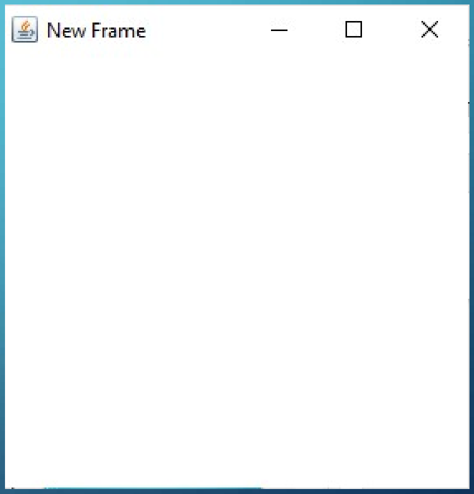
- There are two ways to create a Frame Window.
  - By instantiating Frame class.
  - By extending Frame class.

a) By instantiating Frame class.

- Create Frame window by instantiating Frame class.

Example program for creating a frame

```
import java.awt.*;
class First
{
        public void create()
        {
        Frame f=new Frame("New Frame");
        f.setSize(300,300);
        f.setVisible(true);
        }

}
class sample_frame
{

        public static void main(String args[])
        {
        First obj=new First();
         obj.create();
        }
}
```

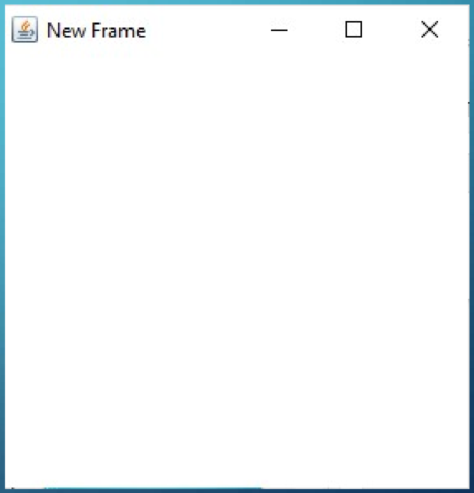


b) By extending Frame class (inheritance):

- We will be inheriting Frame class to create Frame window and hence it won’t be required to create an instance of Frame class explicitly.

Example program for creating a frame by extending Frame class

```
import java.awt.*;
class First extends Frame
{
        First()
        {
        setTitle("New Frame");
        setSize(300,300);
        setVisible(true);
        }

}
class sample_frame
{
        public static void main(String args[])
        {
        First obj=new First();

        }
}
```



AWT Controls

- Controls are components that allow a user to interact with your application in various ways.
- The AWT supports the following types of controls:

  □ Labels

  □ Buttons

  □ Check boxes

  □ Choice lists

  □ Lists

  □ Scroll bars

  □ Text Component

- To include a control in a window, you must add it to the window.
- First create an instance of the desired control and then add it to a window by calling add( ), which is defined by Container.
- To remove a control from a window use the methods remove() or removeall().
- Except for Labels, which are passive controls, all controls generate events when they are accessed by the user.
- In general, our program simply implements the appropriate interface and then registers an event listener for each control that you need to monitor.
- Once a listener has been registered, events are automatically sent to it.

1. Labels

- The simplest form of GUI component o AWT control is the label, which is, effectively, a text string that we can use to label components.
- Labels are not editable; they just label other components on the screen.
- To create a label, use one of the following constructors:
  - Label(String str) creates a label with the given text string, also aligned left.
  - Label(String str, int how):- creates a label with the given text string and the given alignment.
  - The available alignment numbers are stored in class variables in Label, making them easier to remember: Label.RIGHT, Label.LEFT, and Label.CENTER.

2. Buttons

- A push button is a component that contains a label and that generates an event when it is pressed.
- Push buttons are objects of type Button. Button defines these two constructors:
  - Button( ) - creates an empty button.
  - Button(String str) - creates a button that contains str as a label

- Method in Button class are
  - void setLabel(String str) - set the label using the setLabel(String) method.
  - String getLabel( ) - get the value of the button's label by using the getLabel() method.

3. TextField

- TextField is a subclass of TextComponent. It displays a single line of optionally editable text.
- Text field allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.

To create a text field, use one of the following constructors:

- TextField() creates an empty TextField that is 0 characters wide.
- TextField(int numChars) creates an empty text field.
  - The integer argument indicates the minimum number of characters to display.

- TextField(String str) creates a text field initialized with the given string.
  - The field will be automatically resized by the current layout manager.
- TextField(String str, int numChars) creates a text field some number of characters wide (the integer argument) containing the given string.
  - If the string is longer than the width, you can select and drag portions of the text within the field, and the box will scroll left or right.

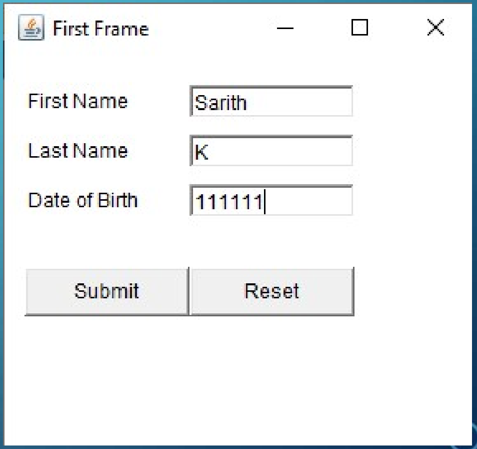
- Eg.

```
TextField tf = new TextField("Enter Your Name", 30);
add(tf);
```

- The above example creates a text field 30 characters wide with the string "Enter Your Name" as its initial contents.

- The methods are
  - String getText() -Returns the text this text field contains (as a string)
  - void setText(String) -Puts the given text string into the field.
  - void setEchoChar(char ch) - method to set the character that is echoed on the screen

```
import java.awt.*;
public class AwtApp extends Frame {
AwtApp(){
setTitle("First Frame");
setSize(300,300);
setVisible(true);
Label firstName = new Label("First Name");
firstName.setBounds(20, 50, 80, 20);

Label lastName = new Label("Last Name");
lastName.setBounds(20, 80, 80, 20);

Label dob = new Label("Date of Birth");
dob.setBounds(20, 110, 80, 20);

TextField firstNameTF = new TextField();
firstNameTF.setBounds(120, 50, 100, 20);

TextField lastNameTF = new TextField();
lastNameTF.setBounds(120, 80, 100, 20);
TextField dobTF = new TextField();
dobTF.setBounds(120, 110, 100, 20);

Button sbmt = new Button("Submit");
sbmt.setBounds(20, 160, 100, 30);

Button reset = new Button("Reset");
reset.setBounds(120,160,100,30);

add(firstName);
add(lastName);
add(dob);
add(firstNameTF);
add(lastNameTF);
add(dobTF);
add(sbmt);
add(reset);
}
public static void main(String[] args) {
AwtApp awt = new AwtApp();

}
}
```



First Name: Sarith
Last Name: K
Date of Birth: 111111

- **setBounds()**
  - The **layout managers** are used to automatically decide the position and size of the added components.
  - In the absence of a layout manager, the position and size of the components have to be set manually.
  - The **setBounds()** method is used in such a situation to set the position and size.
  - To specify the position and size of the components manually, the layout manager of the frame can be **null.**
  - The **setBounds()** method needs four arguments.
  - The first two arguments are **x and y coordinates** of the **top-left corner** of the component, the third argument is the **width** of the component and the fourth argument is the **height** of the component.

The syntax is

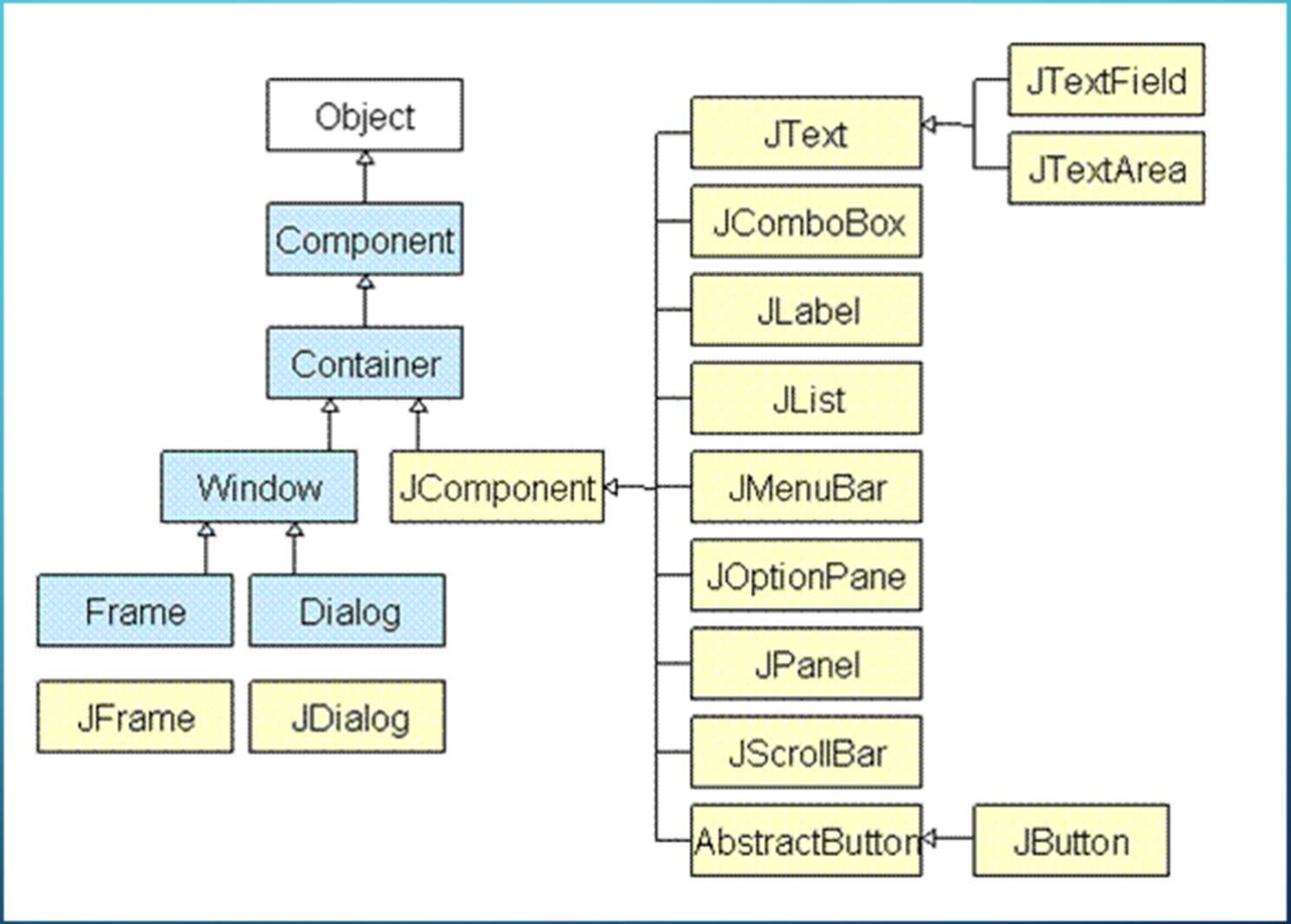
setBounds(int x-coordinate, int y-coordinate, int width, int height)

JAVA SWING

- Java Swing is a GUI Framework that contains a set of classes to provide more powerful and flexible GUI components than **AWT.**
- **Swing** provides the look and feel of modern Java GUI.
- Swing library is an official Java GUI tool kit released by Sun Microsystems.
- It is used to create graphical user interface with Java.
- Swing classes are defined in **javax.swing** package and its sub-packages.

- Java Swing is a part of Java Foundation Classes (JFC) which was designed for enabling large-scale enterprise development of Java applications.
- Java Swing is a set of APIs that provides graphical user interface (GUI) for Java programs.
- Java Swing is also known as Java GUI widget toolkit.

AWT and Swing Hierarchy



Commonly used Methods of Component class

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

JFRAME

- JFrame is a top-level container that provides a window on the screen.
- A frame is actually a base window on which other components rely, namely the menu bar, panels, labels, text fields, buttons, etc.
- Almost every other Swing application starts with the JFrame window.
- Unlike a Frame, JFrame has the option to hide or close the window with the help of the method setDefaultCloseOperation(int).

- Frame class has many constructors that are used to create a new JFrame.
  - **JFrame():** This helps in creating a frame which is invisible.
  - **JFrame(String Title):** Helps in creating a frame with a title.
  -

Creating a JFrame

- There are two ways to create a JFrame Window.
  - By instantiating JFrame class.
  - By extending JFrame class.

a) By instantiating JFrame class.

- Create JFrame window by instantiating JFrame class.

b) By extending Frame class (inheritance):

- We will be inheriting JFrame class to create JFrame window and hence it won’t be required to create an instance of JFrame class explicitly.

SWING COMPONENTS

- **JLabel**
  - The object of JLabel class is a component for placing text in a container.
- JTextField
  - The object of a JTextField class is a text component that allows the editing of a single line text.
- JTextArea
  - The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text.
- JPasswordField
  - The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text.

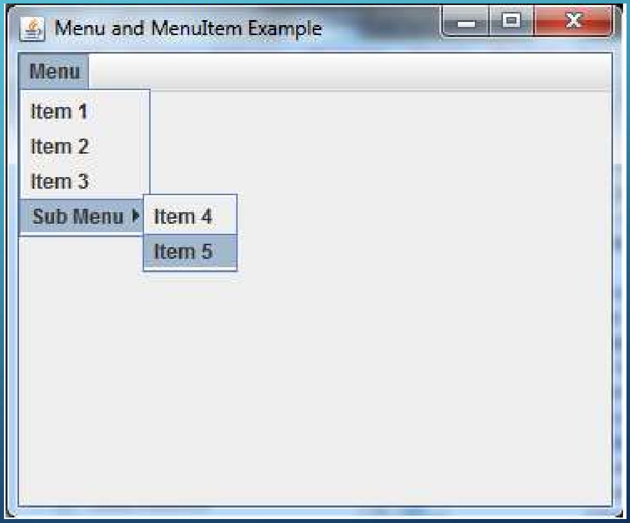
- JCheckBox
  - The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false).
  - Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on “.
- JRadioButton
  - The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options.

- JComboBox
  - The object of Choice class is used to show popup menu of choices.
  - Choice selected by user is shown on the top of a menu.
- JTable
  - The JTable class is used to display data in tabular form.
  - It is composed of rows and columns.

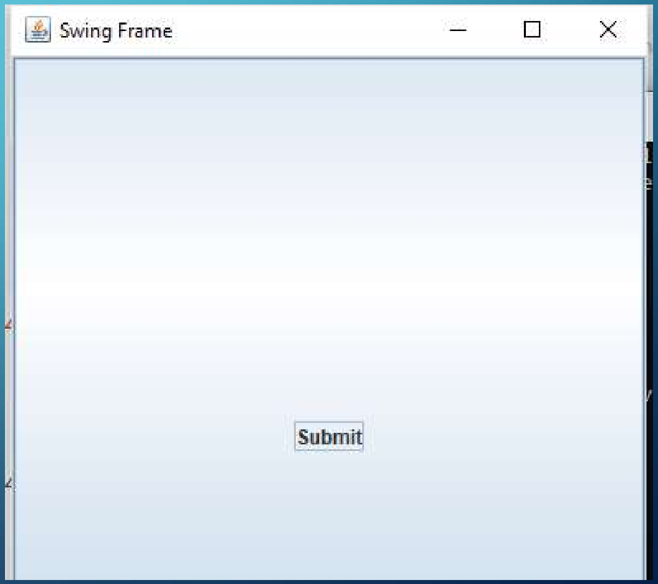
- JList
  - The object of JList class represents a list of text items.
  - The list of text items can be set up so that the user can choose either one item or multiple items.
- JButton
  - The JButton class is used to create a labeled button that has platform independent implementation.
  - The application result in some action when the button is pushed.

- **Java JOptionPane**
  - The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.
  - These dialog boxes are used to display information or get input from the user.

- JMenuBar, JMenu and JMenuItem
  - The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
  - The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.
  - The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

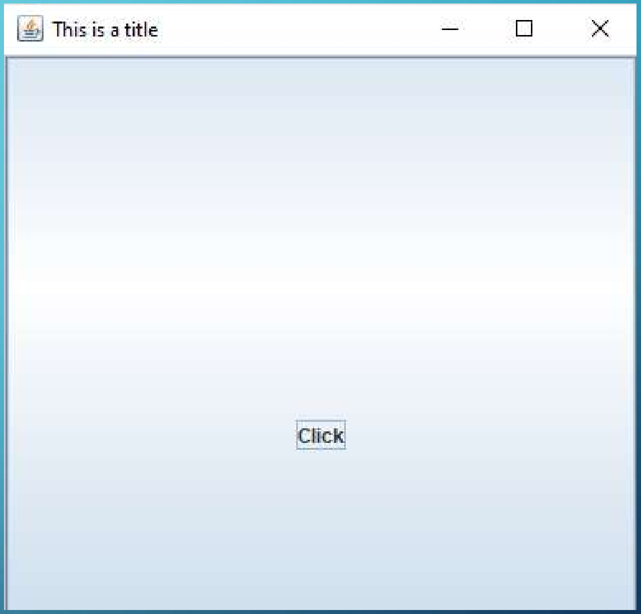


```
import javax.swing.*;
public class SimpleFrm
{
JFrame f;
SimpleFrm(){
        f=new JFrame("Swing Frame");//creating instance of JFrame

        JButton b=new JButton("Submit");//creating instance of JButton
        b.setBounds(130,100,100, 40);

        f.add(b);//adding button in JFrame

        f.setSize(400,500);//400 width and 500 height

        f.setVisible(true);//making the frame visible
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
  public static void main(String args[ ])
{
        new SimpleFrm();
}
}
```



Example Program

```
import javax.swing.*;
public class Simplefrm2 extends JFrame //inheriting JFrame
{
        Simplefrm2()
        {
        setTitle("This is a title");
        JButton b=new JButton("Click");//create button
        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame
        setSize(400,500);
        setVisible(true);
         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
        public static void main(String  args[ ])
         {
        new Simplefrm2();
        }
}
```



Event Classes

- When we press a button in our program or Android application the state of the button changes from ‘Unclicked’ to ‘Clicked’.
- This change in the state of the button is called an Event.
- Events are generated based on how we interact with the GUI.
- The package java.awt.event defines many types of events that are generated by various sources.

- In Java, all events are implemented as classes in the package java.awt.event which form the basis for Java‟s event-handling mechanism.
- When an event occurs, an object of the respective event class is created which encapsulates a state change in the source that generated the event.
- The main Event Classes in java.awt.event is given below

Event class	Description
ActionEvent.	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected
AdjustmentEvent	Generated when a scroll bar is manipulated.
FocusEvent.	Generated when a component gains or loses keyboard focus
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
TextEvent.	Generated when the value of a text area or text field is changed
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces

- The event listeners are the interfaces in java that defines the methods for handling events.
- A listener is an object that is notified when an event occurs. It has two major requirements.
- First, it must have been registered with one or more sources to receive notifications about specific types of events.
- Second, it must implement methods to receive and process these notifications.
- The methods that receive and process events are defined in a set of interfaces, such as those found in java.awt.event package.

- When an event occurs, the source invokes the appropriate method defined by the listener and provides an event object as its argument.
- It is important that the event listener is registered with a source so that any change in the source is notified.

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
TextListener	Defines one method to recognize when a text value changes.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

ActionListener Interface

- This interface defines the actionPerformed( ) method that is invoked when an action event occurs.
- Its general form is shown here:

```
void actionPerformed(ActionEvent ae)
```

- Method to register this interface to the source is

```
addActionListener(ref);
```

Where ref is the currently active source object.

AdjustmentListener Interface

- This interface defines the adjustmentValueChanged( ) method that is invoked when an adjustment event occurs.
- Its general form is shown here:

```
Void adjustmentValueChanged(AdjustmentEvent ae)
```

- Method to register the interface with source is

```
addAdjustmentListener(ref)
```

ItemListener Interface

- Defines a single method that is invoked when the state of an item is changes.
- The syntax is,

  void itemStateChanged(ItemEvent ie)
- Method to register the interface is,

  additemListener(ref);

KeyListener Interface

- This interface defines 3 methods,

  void keyPressed(KeyEvent ke)

  void keyReleased(KeyEvent ke)

  void keyTyped(KeyEvent ke)

- The keyPressed( ) and keyReleased( ) methods are invoked when a key is pressed and released, respectively.
- The keyTyped( ) method is invoked when a character has been entered.

- For example, if a user presses and releases the ‘A’ key, three events are generated in sequence: key pressed, typed, and released.
- If a user presses and releases any functional key (F1), two key events are generated in sequence: key pressed and released.
- Method to register the interface is

```
addKeyListener(ref);
```

TextListener Interface

- Defines a single method that is invoked when a change occurs in a text area or text field.
- Its general form is shown here:

```
void textValueChanged(TextEvent te)
```

- Method to register the interface is

```
addTextListener(ref);
```

WindowListener Interface

- This interface defines seven methods.
- The windowActivated() and windowDeactivated() methods are invoked when a window is activated or deactivated, respectively.
- If a window is iconified, the windowIconified( ) method is called.
- When a window is deiconified, the windowDeiconified( ) method is called.
- When a window is opened or closed, the windowOpened( ) or windowClosed( ) methods are called, respectively.
- The windowClosing( ) method is called when a window is being closed.

- The general forms of these methods are

```
void windowActivated(WindowEvent we)
 void windowClosed(WindowEvent we)
void windowClosing(WindowEvent we)
void windowDeactivated(WindowEvent we)
void windowDeiconified(WindowEvent we)
void windowIconified(WindowEvent we)
 void windowOpened(WindowEvent we)
```

- Method to register the interface is,

```
addWindowListener(ref);
```

MouseListener Interface

- This interface defines five methods.
- If the mouse is pressed and released at the same point, mouseClicked( ) is invoked.
- When the mouse enters a component, the mouseEntered( ) method is called.
- When it leaves, mouseExited( ) is called.
- The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively.

- The general forms of these methods are shown here:

```
void mouseClicked(MouseEvent me)

void mouseEntered(MouseEvent me)

void mouseExited(MouseEvent me)

void mousePressed(MouseEvent me)

 void mouseReleased(MouseEvent me)
```

Method to register the interface is,

```
addMouseListener(ref);
```

MouseMotionListener Interface

- This interface defines two methods.
- The mouseDragged( ) method is called multiple times as the mouse is dragged.
- The mouseMoved( ) method is called multiple times as the mouse is moved.
- Their general forms are shown here:

```
void mouseDragged(MouseEvent me)
void mouseMoved(MouseEvent me)
```

- Method to register the interface is,

```
addMouseMotionListener(ref);
```

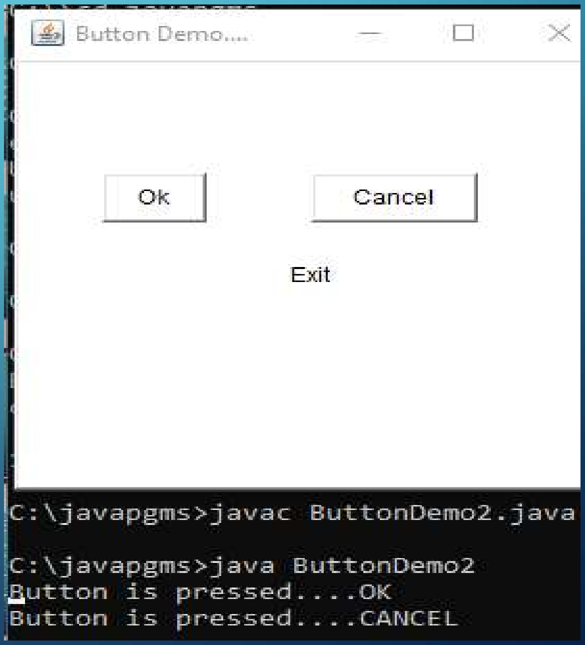
Example program for adding Button inside Frame using AWT

```
import java.awt.*;
import java.awt.event.*;
class CBD implements ActionListener
{
        Button b1,b2,b3;
        public CBD()
        {
                Frame f=new Frame("Button Demo....");
                f.setSize(300,300);
                f.setVisible(true);

                b1 = new Button("Ok");
                b2 = new Button("Cancel");
                b3 = new Button("Exit");

                b1.setBounds(50,100,50,30);
                b2.setBounds(150,100,80,30);
                b3.setBounds(60,150,80,30);
                f.add(b1);
                f.add(b2);
                f.add(b3);
b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);

 }
 public void actionPerformed(ActionEvent ae)
 {
       if(ae.getSource() == b1)
       {
             System.out.println("Button is pressed....OK");
       }
       else if(ae.getSource() == b2)
       {
             System.out.println("Button is pressed....CANCEL");
       }
       else
       {
             System.out.println(0);
       }
 }
}
```

//String str = ae.getActionCommand();

```
class ButtonDemo2
{
      public static void main(String args[])
      {
            CBD obj = new CBD();

      }
}
```

ButtonDemo2.java



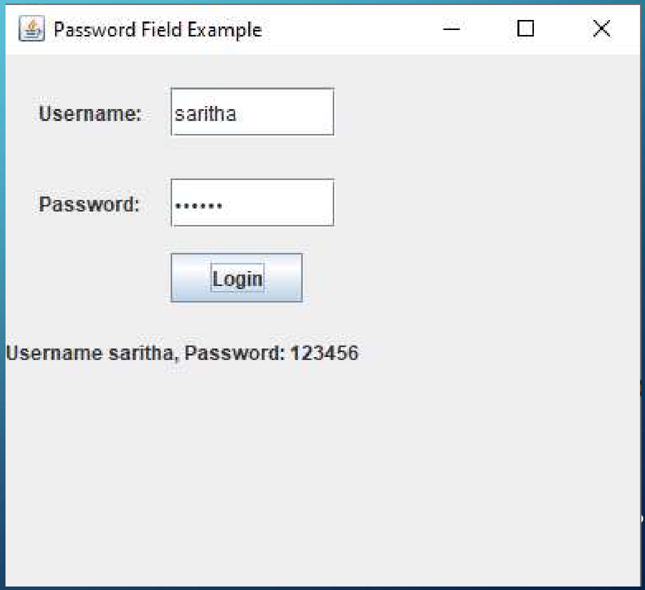
```
C:\javapgms>javac ButtonDemo2.java

C:\javapgms>java ButtonDemo2
Button is pressed....OK
Button is pressed....CANCEL
```

Example 1 pgm using Swing

```
import javax.swing.*;
import java.awt.event.*;
public class ExamplePass implements ActionListener
{
        JLabel label,l1,l2;
        JPasswordField value;
        JButton b;
        JTextField text;
   ExamplePass()
   {
    JFrame f=new JFrame("Password Field Example");
     label = new JLabel();
     label.setBounds(20,150, 200,50);
     value = new JPasswordField();
     value.setBounds(100,75,100,30);
     l1=new JLabel("Username:");
        l1.setBounds(20,20, 80,30);
        l2=new JLabel("Password:");
        l2.setBounds(20,75, 80,30);
```

```
 b = new JButton("Login");
  b.setBounds(100,120, 80,30);
   text = new JTextField();
   text.setBounds(100,20, 100,30);
        f.add(l1);
        f.add(value);
         f.add(l2);
        f.add(text);
        f.add(b);
        f.add(label);

          f.setSize(400,400);
          f.setVisible(true);
          b.addActionListener(this);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

```
public void actionPerformed(ActionEvent e)
 {
          String data = "Username " + text.getText();
          data += ", Password: "   + new String(value.getPassword());
          label.setText(data);
      }
     public static void main(String  args[ ])
{
new ExamplePass();
}
}
```



Username: saritha

Password: ••••••

```
Username saritha, Password: 123456
```

ExamplePass.java

// Perform all arithmetic operations using AWT components

ButtonDemo3.java

JAVA LAYOUT MANAGER

- The Layout manager is used to layout (or arrange) the GUI Java components inside a container.
- The Layout Managers are used to arrange components in a particular manner.
- LayoutManager is an interface that is implemented by all the classes of layout managers.
- A layout manager is an object that controls the size and position of the components in the container.
- Every container object has a layout manager object that controls its layout.
- Actually, layout managers are used to arrange the components in a specific manner.
- It is an interface that is implemented by all the classes of layout managers.

AWT LAYOUT MANAGER CLASSES

- Following is the list of commonly used controls while designing GUI using AWT.

Sr.No.	LayoutManager & Description
1	BorderLayout - The borderlayout arranges the components to fit in the five regions: east, west, north, south, and center.
2	CardLayout - The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3	FlowLayout -The FlowLayout is the default layout. It layout the components in a directional flow.
4	GridLayout - The GridLayout manages the components in the form of a rectangular grid.
5	GridBagLayout - This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally, or along their baseline without requiring the components of the same size.
6	GroupLayout - The GroupLayout hierarchically groups the components in order to position them in a Container.
7	SpringLayout - A SpringLayout positions the children of its associated container according to a set of constraints.

1. GRIDLAYOUT

- Grid Layout is used to place the components in a grid of cells (rectangular).
- Each component takes the available space within its cell.
- Each cell, has exactly the same size and displays only one component.

- In other words, the layout manager divides the container into a grid, so that components can be placed in rows and columns.
- Each component will have the same width and height.
- The components are added to the grid starting at the top-left cell and proceeding left-to-right, until the row is full.
- Then go to the next row.
- This type of layout is known as, the Grid Layout Manager.
-

- There are 3 types of constructor in Grid Layout. They are as following:

1. GridLayout()
2. GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
3. GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

MyGridLayout.java